SENTIMENT, BELIEF AND OPINION DETECTION USING NEURAL NETWORKS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Anusha Chowdhury

May 2018

ABSTRACT

In this work, we explore different techniques to extract opinions, sentiments and beliefs from text. In particular, we look at neural network based approaches since deep learning has gained wide popularity nowadays and is known to perform effectively for the kind of problems we are looking at.

The first goal was to solve the BeSt (Belief and Sentiment) Evaluation task from the 2016 Text Analysis Conference. Here we used bidirectional Long Short Term Memory networks (LSTMs) for sentiment detection in a given document. We looked with particular interest at the sentiment polarity to find out whether it is positive, negative or neutral sentiment. Our neural network based model consists of a multi-layered bidirectional LSTM which takes as input embeddings for the sequence of words in a given sentence along with annotated source and target, and outputs probabilities for the different sentiment polarities. We evaluated the model using standard precision, recall and F1 scores and then compared our results to scores in the existing literature.

The second goal was to identify opinion expressions along with their sources and targets in a given text. We compared a simple baseline approach, support vector machine and conditional random field to a feedforward neural network and a bidirectional LSTM approach. All approaches were evaluated on a standard dataset from the sentiment analysis literature - the MPQA dataset.

ACKNOWLEDGEMENTS

I would like to thank my major advisor Professor Claire Cardie for guiding me throughout this thesis. Without her help, these projects would not have been possible. I would also like to thank her for all the advising office hours, which were very helpful to ensure regular progress and ascertain that we were on the right track. She guided me about how to approach the problem, train the network, improve the model and process the datasets.

I would like to convey special thanks to my minor advisor Professor David Mimno for guiding me throughout these four semesters.

I would like to thank Arzoo Katiyar for helping me with the tensorflow and theano implementations, coding and debugging the models. Her work [1] on opinion extraction using LSTMs was extremely helpful, being related with this project. I had regular interactions with her which helped me a lot to progress fast. She helped with debugging since I was a beginner in tensorflow and was struggling with solving shape errors, handling tensors etc.

I would like to thank Kevin and Zach who were my teammates in the second project on opinion detection. I would also like to thank Vlad and Esin, both PhD students of Claire, for helping me with collection of the dataset and related materials. Also online image sources were very helpful for all the diagrams in this thesis.

I would also like to thank my parents and my friends for being beside me through thick and thin.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Overview

Deep learning is one of the most popular topics in today's world. Various deep learning architectures, such as deep neural networks, convolutional deep neural networks, deep belief networks, and recurrent neural networks, are being applied to fields like computer vision, automatic speech recognition, natural language processing, audio recognition, and bioinformatics [2]. Deep neural networks provide a way of highly automated feature learning and have exhibited considerable potential in solving various tasks in the field of natural language processing, according to online sources. Deep learning comprises algorithms that use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The learning can be supervised or unsupervised. In case of neural network based approaches, latent features are automatically learned as dense vectors of the hidden layers. A nice analogy to the way neural networks function can be found in figure 1.1.

The goal of the first task is to use deep learning to solve the BeSt (Belief and Sentiment) Evaluation task from TAC (Text Analysis Conference 2016). The goal of the second project is to use LSTMs to predict tuples for explicit and implicit opinions in the MPQA dataset (MPQA 2.0 and 3.0). We have outlined different techniques for solving these tasks and compared the results from each of them to existing results.

1.2 Motivation

In recent years, deep learning has been extensively used to solve problems like opinion extraction [1], relation extraction and relation classification (SemEval-2010 Task 8) in the field of natural language processing. Motivated by the recent success of deep learning, we decided to apply deep learning to solve the sentiment and opinion extraction problems here. One can think of deep learning to be like talking over phone [1].

Figure 1.1: Deep learning via neural networks is just like talking over phone or Chinese whisper

Another motivation was the numerous real life applications of this project. For example, extraction of sentiment polarity can help to analyze documents (articles and news) from the world of finance and can help to predict variations in the stock market or to plan marketing and investment strategies. Extraction of sentiment can help get a better overview of movie reviews, book reviews, reviews of any new product in the world of technology etc. Extraction of opinions can help answer questions in several question-answer systems. Nowadays, companies which make video games are using sentiment analysis in their social

[1] https://kidspot.co.nz/activities/spoiled-telephone/

network teams to find out feedback of players about different products from their reviews and chats with other players. They also use sentiment and opinion analysis to suggest potential matches (friends or opponents) for a player. These are just a few examples which convey the fact that this task and related tasks indeed have wide applications in today's world.

It seemed reasonable to explore neural networks and RNNs in particular since they can directly represent essential linguistic structures. Also, we need to capture long term dependencies in the corpus and LSTMs are ideal for that purpose. Previously, feature based approaches (decision trees, random forests and SVMs) have been attempted to solve the BeSt evaluation task. We outline our deep learning based approach and compare the obtained results with those already existing in literature, in the sections below.

CHAPTER 2

THE SENTIMENT AND BELIEF DETECTION TASK

2.1 Task Definition

The Text Analysis Conference (TAC) is organized every year by the Retrieval Group of the Information Access Division in the Information Technology Laboratory at the National Institute of Standards and Technology. The BeSt (Belief and Sentiment) Evaluation [3] is one of the tasks in TAC 2016 involving evaluation of sentiment and belief detection, on a given dataset comprising 246 documents which have annotated sources and targets, where sources are named entities and targets are named entities or events or relations (EREs). The BeSt evaluation is interested in finding sources, attitudes, targets, polarities, and sarcasms. For example, the task tries to detect that in a given document, which entity has what mental attitude towards another entity.

- Source : The source is an entity of type Person, GeoPolitical Entity (GPE), or Organization.
- Target : The target can be any relation or any event. In addition, for sentiment only, the target can also be any entity.
- Evaluation : The evaluation includes both belief and sentiment. The scores are calculated by comparing predicted output from the system to gold annotations. The expected output should contain list of the belief and sentiment relations from entity mentions to entity mentions, relation mentions, and event mentions.

We will look here at the sub-problem which is as follows : given a pair of words, one has to predict whether the polarity is positive, negative or none for sentiment. Once we experiment with the model for this and achieve a good accuracy, we can extend the model to detect belief as well in a very similar manner.

2.2 Background and Related Work

LSTM-RNNs (Hochreiter and Schmidhuber in [4]) have been applied to many sequential modeling and prediction tasks, such as machine translation (Bahdanau et al. in [5]), speech recognition (Graves et al. in [6]) and named entity recognition (Hammerton in [7]). For NER (named entity recognition), the network was trained to perform two passes on each sentence and decisions were made after the second pass. The first pass was used to acquire information which would be used for disambiguation during the second pass. To achieve fine-grained opinion extraction, researchers have focused on extracting subjective phrases using a conditional random field (CRF) based approach from open-domain text such as news articles, as explored by Yang and Cardie in [8]. Deep bidirectional recurrent neural networks have been proposed for identifying subjective expressions (Irsoy and Cardie in [9]), and this gave better performance than the previous CRF-based models. Irsoy and Cardie also proposed a bidirectional recursive neural network over a binary parse tree to jointly identify opinions and opinion entities. In existing literature, LSTMs have been used extensively for joint extraction of opinion entities and relations (Katiyar and Cardie in [1]). The task also involves identifying the IS-FROM and IS-ABOUT relations between an opinion expression and its holder and target. The implementations

for this were done using theano.

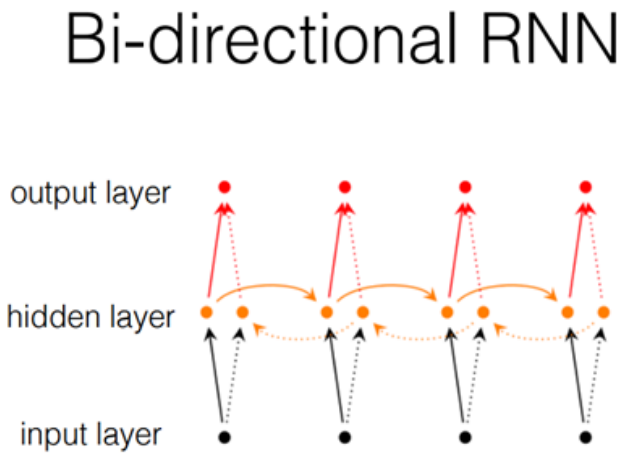

Figure 2.1: Bidirectional RNNs (can have multiple layers) [1]

Relation extraction using LSTMs on sequences and tree structures has been explored in literature (Miwa et al., 2016). Researchers have explored shortest path dependency tree, subtree and full tree and found that shortest dependency paths are most useful when it comes to capturing dependencies, while investigating LSTM models for relation classification (Xu et. al. 2015).

2.3 LSTMs

Recurrent neural networks (RNNs) are connectionist models of sequential data that are naturally applicable to the analysis of natural language (Irsoy, 14). However, it has been seen that in general recurrent neural networks are not good enough to learn long-term dependencies. Long Short Term Memory networks [10] are a special kind of RNN, which are capable of learning long-term dependencies and hence extremely useful in scenarios like belief and sentiment extraction or relation and opinion extraction. The recurrent unit is replaced by

a memory block. The memory block contains two cell states - memory cell state C_t and hidden state h_t and three multiplicative gates - input gate i_t, forget gate f_t and output gate o_t. These gates regulate the addition or removal of information to the cell state thus overcoming vanishing and exploding gradients. Gates are a way to optionally let information through. They consist of a sigmoid neural net layer and a pointwise multiplication operation.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

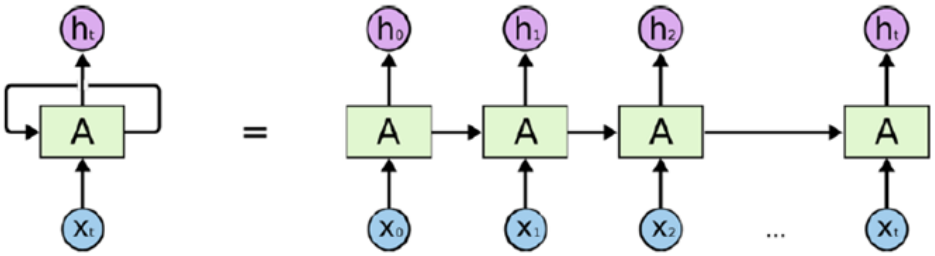

Figure 2.2: Hidden units in LSTM (each unit outputs h and c) [10]

The forget gate and input gate decide which part of the information will be thrown away from the cell state and what new information will be stored in the cell state [1]. The sigmoid outputs a number between 0 and 1 where 0 implies that the information is completely lost and 1 means that the information is completely retained. The intermediate cell state and previous cell state are used to update the new cell state.

$$\tilde{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

2.4 The Dataset

Linguistic Data Consortium is an open consortium of universities, libraries, corporations and government research laboratories. We used the LDC E63 dataset here. The input comprises source text files and ere.xml files which list the entity mentions, relation mentions and event mentions. The documents consist of blog posts, newswires and quotes (labelled as nw and df in the BeSt folder). There are gold annotations (given by experts) and predicted annotations (labels from the system). In other words, gold entities, relations, and events (EREs) and predicted EREs are provided in the dataset. We use 173 text documents for training and 73 for testing. Blog is a single post by an author (multiple sentences). It has author name, date/time and id. Some of them have quotes. The other type of document in the LDC dataset is newswire data. Let us look at an example post by an author izzoh.

"I love my parents to pieces. To cut a long story short, my dad had cancer - given 6 weeks to live. My mum gave up work to look after him. 15 years later, my dad is in remission (thank goodness) back at work and my mum is now staying at home and being looked after. Ever since I started working I have helped out. As I feel a child should when they become an adult and start working."

Here 'I' is an entity, 'gave up work' is an event and 'my parents' is a relation with my as first argument and parents as second argument, 'love' is a positive sentiment with 'I' as the source and 'my parents' as the target. The other relations in this passage are 'my dad' and 'my mum'.

2.5 Private State Tuple

The BeSt evaluation refers to sentiment and belief as private states (TAC, 2016) which is logical because it reflects the mental condition of the entity. The task defines a private state tuple as (source-entity, target-object, value, provenance-list) where:

- Source is either an entity or none. The source is the holder of the sentiment or belief.
- Target is an event or a relation. In case of sentiment task, it can also be an entity.
- Value is either sentiment value (positive, negative) or belief value (committed belief, non-committed belief, reported belief). Committed belief means that the source is convinced that the target is true. Non-committed belief means that the source thinks it is possible or probable that the target is true, but is not certain. Reported belief is when an author reports on the belief of a different source, without letting the reader know what his or her own belief state is.
- Provenance-list is a list of pointers to the text passages which support the identified claim about belief or sentiment. The provenance-list contains an entry for every single piece of textual evidence that supports the specific private state claim expressed by the private state tuple.

Any private state expressed in a document collection can be represented as a collection of private state tuples. A source can have several different private states with respect to the same target (TAC, 2016). For example, the writer can

have positive sentiment towards the election of Clinton, and also have a non-committed belief towards it. A source can even have conflicting private states, for example both positive and negative sentiment. This happens when someone changes his or her mind, or when they react to different aspects of the target.

2.6 Methodology and Experiments

There are two ways to represent relations between entities using neural networks: recurrent/recursive neural networks (RNNs) and convolutional neural networks (CNNs). Our first experiment is using CNN on the MNIST digit classification dataset. This acts like a tutorial for tensorflow implementations and then we propose a RNN-based model for the BeSt evaluation task.

2.6.1 Tensorflow on MNIST dataset

TensorFlow is a powerful open source software library for doing large-scale numerical computation. One of the tasks at which it excels is implementing and training deep neural networks. We chose TensorFlow because it gives faster implementations of neural networks (compared to previous libraries like theano). Being a beginner in neural networks and their implementations in tensorflow, we decided to first do an experiment on a subset of the MNIST digit classification dataset [1].

We referred to the deep MNIST website [11] for the tensorflow implementations in python and then decided to make changes to the input and output as

[1] http://corochann.com/mnist-dataset-introduction-1138.html

required. So we chose a subset of the MNIST dataset consisting of 4000 images, each of which has 784 features (28 X 28 pixels) for training the model and then tested it on a test dataset of 800 images. We built a multilayer convolutional neural network to train and test on this data. We use weights (W) and biases (b) to suit the shapes of these tensors, that is in accordance with the image sizes (784 is to match the input feature vector) and 10 is the number of categories (digits can be 0 to 9). We used argmax method of tensorflow to find the index which had the highest value of probability (hence being the most probable digit) after doing softmax. We also used dropout to prevent overfitting the model.

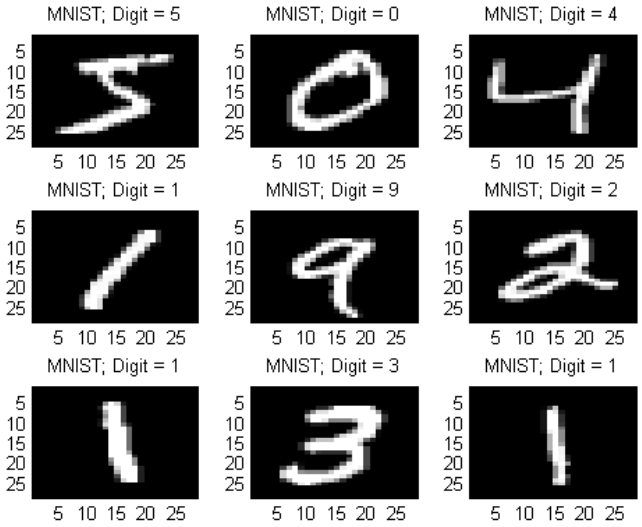

Figure 2.3: Samples from the MNIST digit dataset

Experiments and Evaluations :

We experimented with 20000, 30000 and 40000 iteration sizes and batch sizes of 50, 70, 100 and 110. Out of these, the one with 30000 iteration size and batch size of 70 gave around 98.5% accuracy. With 40000 iteration size and 110 batch size we got 99.25% accuracy. The aim of this exercise was to learn implementations of neural networks in tensorflow. We also observed the fact that batch size and

epochs are important parameters that govern the results, and that there is an optimal spot after which error may increase due to overfitting of the model.

2.6.2 Data Preprocessing

We need to do a token level indexing before we start training the network. We first parsed each document using the Stanford Parser to obtain a token number for each token, sentence wise for the training data starting from 1 (so that 'None' can be marked as token 0) and also get the parts of speech for each token. All these results have been stored in a folder. Then, we create a dictionary from the training vocabulary, so that to refer to any token, we just need its index. We store the dictionary using pickle. UNK is given index 0 and None is given index 1. The total number of words in the training vocabulary is 13230, out of which there are 6526 words whose frequencies are 1. Handling unknown words is always a challenge and we have multiple options to do this. Here to handle UNK, we did the following.

When we get words with frequency of 1 in the training data, we generate a random integer between 0 and 1000, and if this random integer is less than a certain threshold then we treat it as an unknown word (as if it is seen for the first time, just like UNK). We found that setting this number as 5 gives us around thirty UNK tags. We experimented with this parameter further to check at which point accuracy is the maximum and we found that around 7 we get the best result. This parameter setting gives around fifty UNK tags on average. For feeding each document into the LSTM we create the following:

- The indices corresponding to each token in the document are stored in a

list. We also store the maximum sequence length as a list.

- The different pairs (combination of tokens) are stored as list of lists, where we now just need the beginning and ending indices for each pair.
- The labels corresponding to each pair are stored in a list.

All these (that is input token indices, maximum sequence length, pairs and labels) are fetched each time we call a batch for feeding into the LSTM.

2.6.3 Word Embeddings

Distributional Hypothesis, as per online definition, states that words that occur in the same contexts tend to have similar meanings. We use Word2Vec embeddings here, more specifically the gensim word2vec model. This returns a 300×1 embedding for each given word. Even UNK has a representation as per *gensim.models.keyedvectors* which was really helpful. Initially, we used Google News corpus, but then that was not suitable for the dataset we are dealing with from TAC since we have posts, quotes, newswires etc. So we used a bigger LDC dataset from NIST website in order to resemble the discourse of the dataset we are dealing with here. The LDC E63 English dataset was in the form of xml files, so we used xml.etree (element tree) parser to get rid of the xml tags and convert it into a text document format, as required by LineSentence package of gensimword2vec. The model from LineSentence gives 100×1 embeddings and unlike the previous model, this one does not have any default embedding for unknown words. We have used a window size of 5 and mincount has been set to 1 so that even words which occur just once are captured by the model.

2.6.4 Hypothesis

Our hypothesis is that deep LSTM networks should perform well for sentiment detection, given properly trained word embeddings (which capture the semantics of the tokens, being trained on similar discourse). Also we feel that epochs and the unknown word parameter would be playing a key role in deciding the performance of the model.

2.6.5 The Model

We cannot use the CNN model of MNIST for the BeSt task because here we are dealing with word pairs and sentiments, so we need context. Hence, we propose a multi-layer bidirectional LSTM model as follows. The model has the following layers : word embeddings layer, bidirectional LSTM layer, feedforward network layer and output layer as shown in figure 2.4. By stacking multiple layers, we incorporate the notion of depth in our network. The bidirectionality ensures that information is incorporated from preceding as well as following tokens at any point.

In the first layer which we refer to as the word embeddings layer, the word2vec representation of tokens in sentences from the documents is fed into the model. The output expected finally from the network is 0 (none), 1 (positive sentiment polarity) or 2 (negative sentiment polarity). This layer takes 300×1 word2vec embeddings corresponding to each token as input (where we were using the Google News trained model earlier and then we trained the word embeddings on the LDC E63 corpus) and the rnn-output is of dimension of hidden size which is 50×1 here. This is the embeddings layer (as shown in figure 2.5 below).

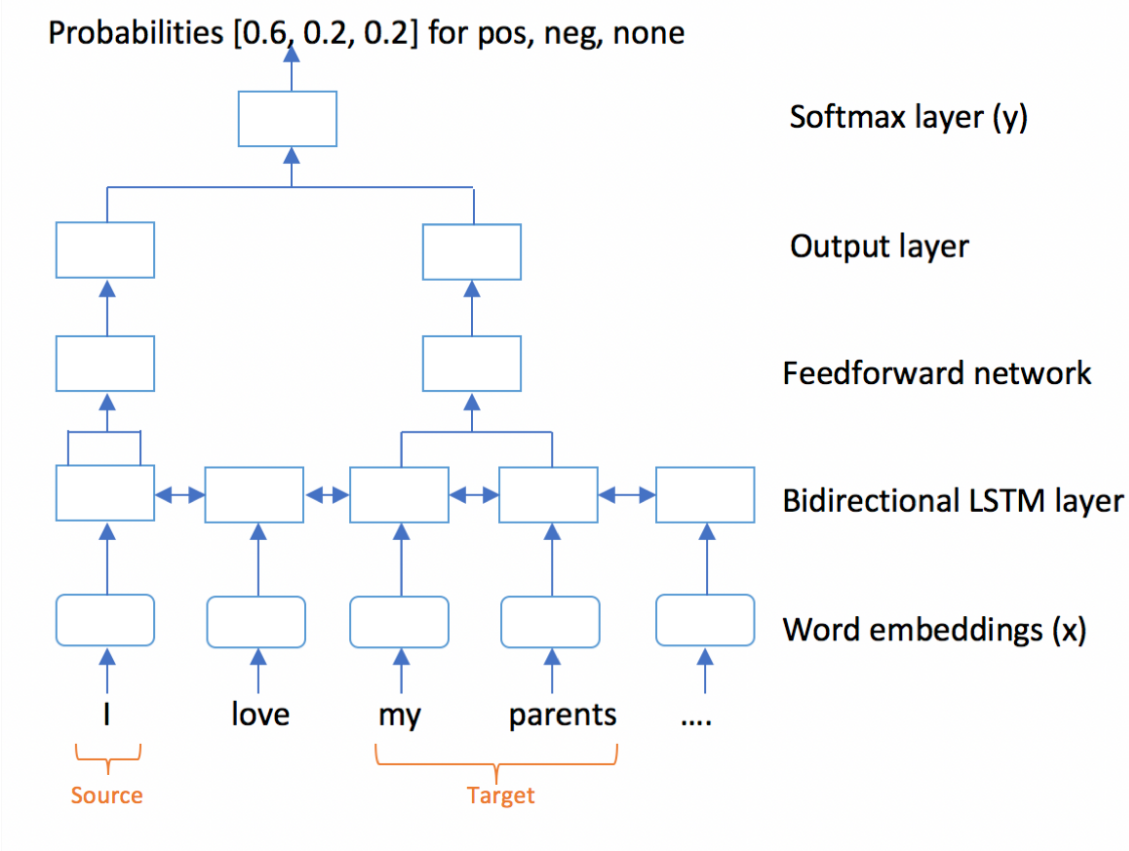

Figure 2.4: Our LSTM based model

Signal for pairs is given in the second layer by concatenating the respective outputs (corresponding to pairs) in the signalling layer. The feedforward network is used for changing dimensions from hidden size to output size. The output layer outputs a 3×1 vector containing the probabilities for each class (probability distribution obtained from softmax). Finally to get the prediction as output, we need to take the index of the maximum element in this vector. In order to measure loss, we use cross entropy (loss calculation is important because the network learns from back propagation).

We used entity pretraining and shared parameters because these seemed to give reasonable improvements in performance in existing literature (Miwa,16).

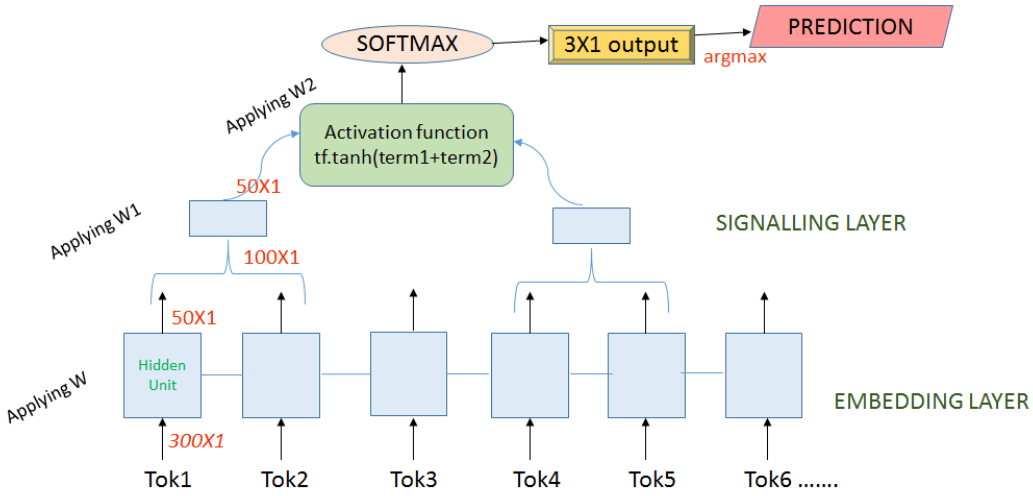

Figure 2.5: Workflow of our LSTM based model

The notion of shared parameters is that when learning from backpropagation happens, all the parameters (weights and biases across all layers) are updated. Dropout strategies were taken in order to avoid the problem of overfitting. By randomly omitting a fraction of the feature detectors from the network during training, it can achieve better performance.

2.6.6 Different Training Methods

We first trained a neural network where we fed the entire document to the LSTM because we thought sentences will help capture the context and hence it would be reasonable to give as much context as we can. However, after careful consideration, we found that this may be giving it too much context, because as per existing literature [1], we have found that the context is most relevant generally when the distance is small (like words close to the pair that we are considering) and tends to decrease as we increase the distance (in terms of number of tokens from the source or target). So, we decided to just feed in the vector representa-

tions of tokens for the pair under consideration (the source and the target) and for the words lying in between them. Another thing we explored is whether there is any considerable improvement in performance if we train one network on blog posts and another network on newswires, that is separating these two categories. However the concern there is that the number of documents available for the newswire category are too less to train a neural network properly. So, we used all the documents available to us for training the network.

2.6.7 Implementation Details

The implementation of the model has been done in python. For parsing the xml files, we used the etree module from lxml. For the model, Tensorflow seemed to be a good choice as it is a brand new open source software library and quite popular too. We import the following modules from tensorflow: contrib.rnn , seq2seq, core rnn cell , array ops and nest. We defined placeholders for each of the parameters, defined getBatch to load a batch on each iteration and when the session runs, we feed in values to each of these placeholders to train our network. We used the DropoutWrapper module of contrib.rnn for our dropout strategies. Finally we feed in the pairs from the test documents, the model returns us the predictions and then we compare it to gold standards to check the accuracy, precision, recall and F1 scores. This is just a very short description of the main script. We also had numerous other scripts for data preprocessing like using pickle to dump the dictionary, using parsers to get all the details about tokens (count, POS tags), loading word2vec models etc.

Looping through tensors was one of the major challenges and took a long

time. We had to read the documentations for different functions on the tensorflow library in order to find out which one would suit our needs. The other major challenge was debugging the code. We faced issues like shape errors with tensors, index errors because of maximum sequence length limit (which we then increased accordingly), loop errors etc. And after a lot of debugging, we finally got a running system. There were a few methods which we found really helpful while training the networks. One of them is tf.nn.embedding lookup which looks up ids in a list of embedding tensors. We had two variations for our model, one was feeding in the entire document and the other one is to feed in the words that occur in between the pairs that we are looking at. Now to perform this, we need to loop through the output from the hidden units.

Our first attempt was using tf.SparseTensor followed by tf.pack or tf.stack but the problem in this case is that the loop variable keeps on changing shape (as pairs can consist of variable number of tokens) and that kind of loop is not allowed in SparseTensor. Also, tf.assign did not help because initial value of tensor variable must have a shape specified, and the number of words are not known beforehand. Finally, we used inbuilt methods of nest from tensorflow.python.util package.

2.6.8 Hyperparameters and Training Details

Selection of hyperparameters is one of the essential tasks when dealing with deep learning. The different hyperparameters that we had to specify for this LSTM model were maximum epoch, batch size, hidden size, number of layers, learning rate, training fraction, dropout, maximum sequence length and steps

per checkpoint (please refer to config.ini in the attached code folder on github). Among these we kept the hidden size always as 50, number of layers as 3, learning rate as 0.01, batch size as 1, steps per checkpoint as 10 (whenever the iteration is a multiple of this, we run our network on the test data) and maximum sequence length as 5000.

For the rest of the parameters we experimented with different values. Ideally, we should be experimenting with each of these parameters, but as we have too many, due to time constraints we decided to restrict ourselves to a few of them. We used the standard multi-class cross-entropy as the objective function when training the neural networks. We used stochastic gradient descent with a fixed learning rate (0.01) and we update the weights after every batch. We regularize our network using dropout.

2.7 Evaluation

We use precision, recall and F-measure for performance evaluation. We experiment with the proposed model by changing epoch size and changing number of tokens treated as UNK. We report F1 scores and accuracy values in the table below. The previous implementations done in theano had a similar F1 score. We also calculated the precision and the recall and the problem with those was that precision came out to be lower than expected (around 0.14) and recall was higher than expected (around 0.8) which implies that the LSTM model is giving us some false positives, because the actual sentiment is none and it is predicting positive or negative in some of these cases as per the softmax probabilities.

One limitation of this model is that we cannot increase the epochs beyond a

Participating Team	F1 score
Baseline	0.145
Columbia	0.206
CornPittMich	0.195
CUBISM	0.151

Table 2.1: Existing F1 scores

Parameters	F1 score	Accuracy
100 epochs and unk = 5	0.23	36.7%
150 epochs and unk = 7	0.24	39.1%

Table 2.2: Obtained F1 scores

few hundreds because then tensorflow runs out of memory. Neural networks being very slow in general, it takes lot of time to train the model. Please note that the table reports only three of our experiments but actually we did perform many more, however these were the ones which gave good F1 scores. When the epoch was set to too low like around 4 to 7, the accuracy was very low (around 20% and hence not reported in the table) which makes sense because the model needs enough iterations to train. When the epoch value was too high like 300, tensorflow suffered from memory error, which is a problem with tensorflow models in general. So we report the ones which were significant in our experiments.

CHAPTER 3

THE OPINION EXTRACTION TASK

3.1 Task Definition

As evident from the section on motivation, people want to detect explicit and implicit opinions toward entities and events given a block of text. Our goal here is to be able to predict the opinion tuple (source, opinion expression, target, polarity) given a sentence from the MPQA corpus. This is the basic four-element tuple representation for any opinion, one can also add more components like sarcasm, degree of polarity etc. We do several experiments on the MPQA 2.0 and 3.0 datasets to detect opinions and then evaluate our results using precision, recall and F1 scores. We initially attempt a baseline approach followed by techniques including SVMs, CRFs with ILP, feedforward neural nets and LSTMs.

As an example, consider the sentence "When the Imam (may God be satisfied with him) issued the fatwa against Salman Rushdie for insulting the Prophet (peace be upon him), the countries that are so-called supporters of human rights protested against the fatwa." taken from MPQA dataset and mentioned in [12]. In the first clause, the writer is positive toward Imam and Prophet. In the second clause, the writer is negative towards the countries and the countries are negative towards fatwa. Thus, positive opinions are (writer, Imam), (writer, Prophet) and negative opinions are (Imam, Rushdie), (Imam, insulting), (Rushdie, Prophet), (writer, countries) and (countries, fatwa).

3.2 The Dataset

The annotations in the MPQA (Multiperspective Question Answering) data collection are copyrighted by the MITRE Corporation. In this work we use versions 2.0 and 3.0 of the MPQA dataset [13].

The MPQA 2.0 corpus contains news articles and other text documents manually annotated for opinions and other private states (that is beliefs, emotions, sentiments, speculations etc.) The corpus contains 692 documents with a total of 15802 sentences.

The main changes in the 3.0 version of the MPQA corpus are the additions of new eTarget annotations. Previously, the target annotations in MPQA 2.0 are spans. In MPQA 3.0, the eTarget annotations are entities and events (entity/event-level target), which are anchored to the head words of noun phrases or verb phrases. This corpus comprises of 70 documents and has the following annotations as per [13] :

- Agent annotation : Agents are sources of the private states. Every agent is given an id. The nested-source is a list of agent ids beginning with the writer and ending with the id for the immediate agent being referenced.
- Expressive-subjectivity annotation : Expressive-subjective elements are words and phrases that indirectly describe a private state. For example, in the sentence 'We foresaw electoral fraud but not daylight robbery', the words fraud and robbery are expressive-subjective elements that indirectly express a private state.
- Direct-subjective annotation : Direct-subjectives are direct mentions of private states and speech events (spoken or written) expressing private states.

For example, in the sentence 'It was unrealistic to envisage that people who were suffering from lack of development, poverty, hunger and poor health could fully enjoy their human rights.', the words 'lack', 'poverty', 'hunger' and 'health' are direct-subjectives.

- Objective-speech-event annotation : These are speech events that do not express private states. They have an id, a nested source and a target frame.
- Attitude annotation : These are the attitudes that compose the expressed private states. They have an id and a type such as positive sentiment, negative arguing, disagreement etc.
- Targetframe annotation : Records the span-based target annotations and entity/event-level annotations for each attitude, expressive subjectivity and objective speech event. This was not present in the 2.0 version.
- starget annotation : This marks the span-based targets of the attitudes, that is what the attitudes are about or what the attitudes are directed toward.
- etarget annotation : This marks the entity/event-level target of the attitudes, expressive subjectivities and objective speech events. The eTarget is anchored to a noun phrase head or a verb phrase head. This annotation is added in the MPQA 3.0 version.
- Sentence annotation : This marks each sentence.

3.3 Data Preprocessing

We used xmltree in python to parse the xml files from MPQA dataset containing the annotations. We looked at expressive subjective elements (ESEs) and

direct subjective elements (DSEs). We treat opinion as the tuple (Agent, Opinion expression, Target(s/e), Polarity) and to find keywords from any expression, we used the Rake algorithm (Rapid Automatic Keyword Extraction algorithm) with NLTK. We extract the source, target and polarity for the DSEs and ESEs, and store them in a tab-separated format along with the sentences in which they occur.

3.4 Existing Literature

As mentioned earlier, our goal is to do opinion extraction. In existing literature, Lingjia et. al. explored opinion inference from a non-neural network approach and Pichotta et. al. explored neural networks and script learning for doing event inference. The thesis of Lingjia Deng was helpful to generate ideas for solving our task.

In [12] Deng et. al. builds an entity/event-level sentiment analysis system which is able to recognize and infer both explicit and implicit sentiments toward entities and events in a given piece of text. Our ideas for the baseline approach and SVM were inspired by [12]. Probabilistic Soft Logic (PSL) models were applied for entity/event-level sentiment analysis. Each logical rule has a corresponding weight, and each predicate has ground value or score associated with it when values are substituted. The paper outlines three span-based sentiment analysis systems which are as follows. S1 extracts triples of <source span, opinion span, target span> but does not extract opinion polarities. S2 extracts opinion spans and opinion polarities but it does not extract sources or targets. S3 extracts opinion spans and polarities. The paper also talks about three ways

to select etarget candidates. ET1 considers all the nouns and verbs in the sentence to provide a full recall of eTargets. ET2 considers all the nouns and verbs in the target spans and opinion spans that are automatically extracted by systems S1, S2 and S3. ET3 considers the heads of the target and opinion spans and their siblings that are extracted by systems S1, S2 and S3.

[14] provides an annotation scheme for adding event and entity target annotations in the MPQA corpus, that is adding etarget annotations within a target (this is the basic difference between MPQA 2.0 and 3.0). Etarget is generally the head word of a noun phrase or verb phrase that is included in the target of the sentiment. For example, in the sentence ”Three leading international organisations warned jointly Thursday that the international fight against terrorism should not be a pretext for the violation of human rights.” the target is the entire phrase ”the international fight ... rights” but the etargets are ’be’, ’pretext’ and ’violation’. One possible application of this work is automatic question answering, since more granular annotations implies that more sentiment relationships can be extracted.

In [15] Deng outlines her thesis proposal which is very closely related to our topic here. The inference rules mentioned in [12] are used to develop an entity/event-level sentiment analysis system that aims to detect both explicit and implicit sentiments expressed among entities and events in the text.

For opinion inferences and event inferences, the first step is to detect polarity. A gfbf is an event that is goodFor or badFor some entity/object as explained in [16]. Any event is represented as a contiguous triple of text spans: (agent, gfbf, object). An influencer is defined as a word that either retain or reverse the polarity of a gfbf event. For example, in the sentence ’The reform pre-

vented companies from hurting patients.' the influencer is 'prevented' because it changes the polarity from badFor to goodFor. These concepts were outlined in [16] for event inferences but can be easily extended to opinion inferences in a similar manner.

In [17] Bishan et. al. explores opinion entity identification and opinion relation extraction (relation can be of two types : IS-FROM and IS-ABOUT) using CRFs. They also look at opinion-argument relations and opinion-implicit argument relations.

[18] defines event as a tuple (v, s, o, p^*), where v is a verb, s is a noun standing in subject relation to v, o is a noun standing as a direct object to v, and p^* denotes an arbitrary number of (pobj, prep) pairs, where pobj is a noun related to the verb v via the preposition prep. For example, the event 'Napoleon sent the letter to Josephine' can be represented by the tuple (sent, Napoleon, letter, (Josephine, to)). Scripts are defined as structured models of stereotypical sequences of events. In this work, LSTMs were trained using various encoder-decoder models for comparison. Each of these models first encodes a sentence (either its events or text) into a learned hidden vector state, and then decodes that vector into its successor sentence (either its events or its text). Text prediction models encode a sentence's raw words and then decodes the predicted sentence's words. Event inference models encode a sentence's events and decodes the first predicted event.

Yet another way is to train statistical models for co-occurring events as described in [19]. A count-based event co-occurrence system is proposed. One event component is given as input per time step and the method does a beam search over the components of the next event to be inferred. Also low dimen-

sional word embeddings are used to capture the generalization beyond the lexical level. For example, this will allow verbs like 'flew' and 'soared' to get similar representations.

There are (among others) two standard training/testing setups using Recurrent Neural Networks : the RNN sequence model and the RNN sequence-to-sequence model. In the basic setup, the input at a timestep is a one-hot vector identifying the word, and the output is a distribution over next tokens, trained to predict the next timesteps input.

In [20] Pichotta mentions two training/testing setups using recurrent neural networks : the RNN sequence model and the RNN sequence-to-sequence model. As per the basic setup, the input at a time step is a one-hot vector identifying the word and the output is a distribution over the next tokens, which is trained to predict the next time step's input.

[21] outlines two script model approaches. Noun model approach learns predicting events as verb lemmas, noun lemmas and prepositions. Entity model approach learns verbs, entity IDs and prepositions, where entity ID is an integer which identifies an argument's entity according to a coreference resolution engine. The system was evaluated on two tasks - inferring held-out events from text and inferring novel events from text.

3.5 Baseline Approach

GOAL : The goal is to predict whether a given word is an etarget or not.

METHODOLOGY : We label each noun and verb as an etarget.

EVALUATION : We calculate precision, recall and F1 scores using the gold standard annotations for MPQA 3.0. As expected, the precision is low because of false positives and recall is very high since average probability of complete retrieval is high. Our results are outlined in table 3.2 below. These results were very close to what Deng et. al. had obtained from their experiments in [12], which is shown in table 3.1 below.

Category	Precision	Recall	F1
Positive polarity	0.1481	0.4857	0.2270
Negative polarity	0.1824	0.6408	0.2840

Table 3.1: Results of predicting etarget from [12]

Category	Precision	Recall	F1
All polarities	0.1614	0.9049	0.2739

Table 3.2: Our Results of predicting etarget from baseline approach

3.6 Support Vector Machines

We explored SVM approach to solve two tasks : predicting etargets given a word and predicting polarity given an opinion.

3.6.1 Predicting Etargets using SVM

GOAL : The goal is to predict whether a given word is an etarget or not.

METHODOLOGY : The feature vector X contains a set of vectors corresponding to each word. These features include the following:

- Distance of the word from the source
- Word embedding (Word2Vec or GloVe)
- Whether it is part of an opinion span
- Unigrams from the given word to opinion word in dependency parse graph
- POS (parts of speech) tags of the words

The label y can be 0 or 1 indicating whether this noun or verb is an etarget or not. This approach was tried on the MPQA 3.0 dataset. There were 15641 data points (feature vectors). We used 80 percent of the data for training and 20 percent for testing. The SVM tends to predict the most common label which is 0 (since most words are not etargets) if the entire input data is used for training. Hence, we experiment with samples of different sizes. We use opinion polarity to get two different varieties of samples. For implementation, we used the svm method available in sklearn package of python. We also gave a balanced set of training data, meaning that it had sufficient points from each class (class zero and class one). In order to give one hot vectors, we need to input a vector of size 15641 which is a very high dimension for svm. Hence we decided to use word2vec embeddings (100 X 1 dimensional). Also, the distance feature did not improve results in any significant way.

EVALUATION : When we use a training set of length 2278 and a test data set of length 456, we get 363 true positives, 0 false positives and 0 true negatives

and 93 false negatives. The accuracy was 0.796 approximately.
When we use the entire training set, we get 2927 true negatives, 214 false negatives and no true positives at all.
With positive polarity etargets, a training dataset of length 450 and test dataset of length 90 gives 85 true positives, 0 false positives, 0 true negatives and 5 false negatives. The accuracy was 0.94 approximately.
With negative polarity etargets, a training dataset of length 882 and test dataset of length 177, we get 155 true positives, 0 false positives, 0 true negatives and 22 false negatives. The accuracy was 0.876 approximately.
Here are some samples from the correct classifications we got using SVM.

- Sentence : Some Westerners who have been there have also seen the ever human rights in the Tibet Autonomous Region, he added.
  Correct classification : 'rights' was correctly classified as etarget by our svm, with positive polarity.
- The United Nations has held the high goal of protecting human rights, Jin said.
  Correct classification : 'goal' classified as etarget with positive polarity.
- Put simply, what we are seeing is the naked pursuit of US interests.
  Correct classification : 'pursuit' classified as etarget with negative polarity.

3.6.2 Predicting Polarity using SVM

GOAL : The goal is to predict the polarity of a given opinion. This section was done jointly with Kevin. Achieving the goal would enable us to automatically produce the positive and negative opinion pairs for a given sentence, where an

opinion pair corresponds to a (source, etarget) tuple.

METHODOLOGY : The interesting aspect here is the way to generate answer candidates. One could consider all (noun, verb) pairs and (noun, noun) pairs in a sentence as answer candidates. One could also restrict sources using named entity recognition or consider the head words of the noun phrases and verb phrases. Another way is to depend on the systems S1, S2, and S3 mentioned in [12]. S1 extracts source span, opinion span and target span, but does not extract opinion polarities. S2 extracts opinion spans and opinion polarities, but it does not extract sources or targets. S3 is trained on movie review data, and extracts opinion spans and polarities.

80% of sentences in MPQA 3.0 were used for the training set and 20% of sentences used for test set. The following features were used:

- Embedding for the sentence (average of the embeddings of the words)
- Embedding for source/agent (average of the embeddings of the words)
- Embedding for the eTarget

Each of these word embeddings is obtained from pre-trained GLoVE embeddings (50-dimensional) and then all of these three are concatenated together to represent any tuple. The output is the polarity of the tuple (positive, negative or neutral).

EVALUATION : The results obtained are given in table 3.3. The gold standard count of total Pairs in positive set and negative set is 536. The gold standard candidate overlap is 430 for the second strategy and 353 for the third strategy.

The average number of candidates per sentence is 39.3 and 27.6 in these cases respectively.

Candidate Generation Strategy	Polarity	Precision	Recall	F1 score
All (noun, verb) pairs and (noun, noun) pairs in a sentence as answer candidates	Positive	0.023	0.044	0.03
Same as above	Negative	0.045	0.122	0.065
NER library used to restrict sources to be words that correspond to certain named entities and for each source create (source, noun) and (source, verb) pairs for all nouns and verbs that follow the source in the sentence	Positive	0.042	0.086	0.056
Same as above	Negative	0.075	0.192	0.108
NER library used to restrict sources to be words that correspond to certain named entities, and for each source create (source, noun) and (source, verb) pairs for all head nouns and head verbs of phrases in the sentence	Positive	0.037	0.051	0.043
Same as above	Negative	0.065	0.091	0.076

Table 3.3: Results of predicting opinion polarity using SVMs

3.7 Conditional Random Fields

In order to predict the source and target of the opinion in a given sentence, one can use the standard BIO tagging scheme (begin, inside and outside). As we know, CRFs are ideal in scenarios where one is dealing with a sequence labelling task. Hence, we decided to explore CRFs for this problem.

GOAL : The goal is to predict a tag for each word in a sentence in order to

identify the source, target and opinion expression. The tag can be any of the following : begin_source, inside_source, begin_DSE, inside_DSE, begin_target, inside_target and outside. In addition to that, the goal also involves predicting relations jointly. The relation is either of type IS-FROM (between source and opinion expression) or IS-ABOUT (between opinion expression and target). An example of the tagging we want to achieve is explained in figure 3.1.

METHODOLOGY : The linear chain CRF predicts sequences of labels for sequences of input samples, based on the principle of maximum likelihood learning. Moreover, we need to take into account the predictions for the surrounding words when we are predicting etargets and a CRF is useful in such cases where one needs to take context into account. As shown in figure 3.2, a linear chain CRF learns the parameters through maximum likelihood learning, equations for this have been obtained from [22]. We referred to the experiments outlined

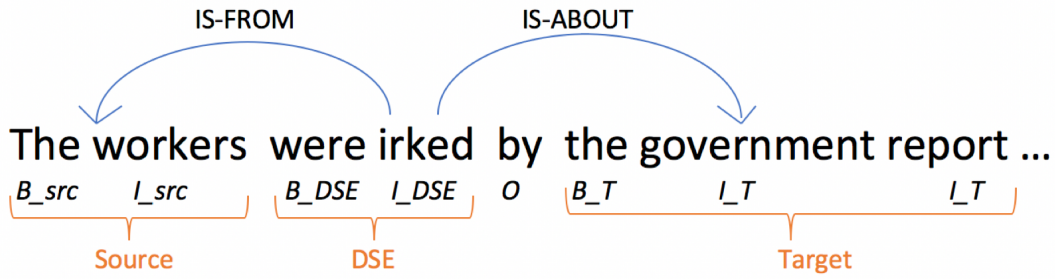

Figure 3.1: An example of joint prediction of entities and relations

in [17] where Bishan et. al. uses BIO tagging for agents and targets in the MPQA dataset. The feature vector consists of word embeddings and POS tags. Entity identification is treated as a sequence tagging problem and relation extraction as a binary classification since relation can be either IS-FROM or IS-ABOUT. For

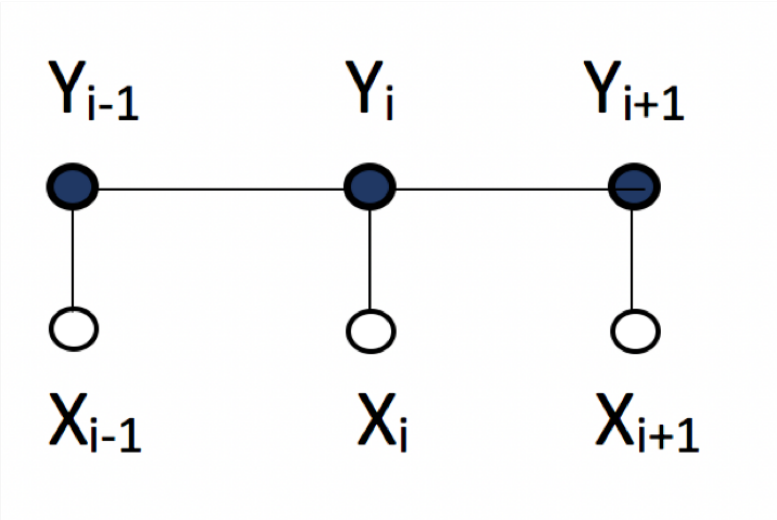

Figure 3.2: The Linear Chain CRF for sequences

joint inference of entities and relations, the GLPK package was used to provide the most optimal ILP solution.

EVALUATION : There were 315 training documents and 35 test documents. We ran experiments with a set of 6281 training sentences and 892 test sentences from MPQA 2.0, and evaluated our results based on precision, recall and F1 scores. There can be two types of matches between gold annotations and predicted annotations : exact and proportional. For exact match, the entire sequence has to match. For proportional match, there will be some non-zero overlap between the predicted and the gold annotation. We compare our results to the ones obtained from the CRF+ILP approach outlined in [17] and this comparison is shown in figure 3.3 below. Most of these results are also comparable to the ones mentioned in [1].

EXAMPLE : In the sentence ”Energy Secretary Ernesto Martens has admitted that it is unfeasible to guarantee the flow of revenues to the government if

the Finance Secretariat maintains its estimate that the crude export platform in 2002 will average 1.825 million barrels per day at a price of $17 per barrel." this method predicted 'Energy Secretary Ernesto Martens' as agent, 'has admitted' as DSE and 'the flow of revenues to the government' as target with the relation between agent and DSE being IS-FROM which is correct. It also identified another opinion tuple in this sentence where 'the Finance Secretariat' is the agent, 'maintains its estimate' is the DSE and 'the crude export platform in 2002' is the target and the relation between DSE and target was predicted to be IS-ABOUT which is correct.

Category	Match Type	Precision		Recall		F1	
Agent	proportional	0.7232	0.6316	0.4909	**0.6588**	0.5848	**0.6449**
	exact	0.6518	0.5366	0.4424.	**0.5673**	0.5271	**0.5515**
DSE	proportional	0.8221	0.6423	0.6615	**0.6819**	0.7331	0.6615
	exact	0.6660	0.5365	0.5257	**0.5672**	0.5876	0.5514
Target	proportional	0.7322	0.5716	0.4858	**0.6789**	0.5841	0.6206
	exact	0.4444	**0.4893**	0.2960	**0.5902**	0.3554	**0.5351**
IS-FROM	exact	0.6497	0.6211	0.5861	0.4416	0.6163	0.5161
IS-ABOUT	exact	0.6438	0.5892	0.5120	0.4587	0.5704	0.5158

Figure 3.3: Comparison of results with previous models (The values in left column are from [17] and in right column are from our model)

3.8 Feedforward Neural Network Approach

GOAL : The goal is to predict the polarity given the source and the target of the opinion.

METHODOLOGY : This approach is similar to the SVM based approach in terms of the candidate generation, and also the representation of inputs. This section was done jointly with Kevin. The feedforward neural network components include the following :

- Input layer (Embedding of sentence, Embedding of source, Embedding of eTarget)
- Two ReLU (rectified linear unit) hidden layers
- Dropout (used for regularization)
- Output layer (a vector of three elements corresponding to positive, negative and neutral)

Cross entropy error was used as the loss function. In deep learning, the choice of the optimization algorithm is a key factor which governs the results. The Adam optimization algorithm is an extension to stochastic gradient descent [23]. We used Adam algorithm as the learning algorithm here because it is computationally efficient, straightforward to implement and appropriate for the type of problem we are dealing with.

EVALUATION : For the positive polarity set, the precision was 0.056, recall was 0.025 and F1 score was 0.035 approximately. For the negative set, precision was

0.103, recall was 0.05 and F1 score was 0.067 approximately. Please note that this was a new method we explored and hence there are no previous results to compare to.

3.9 LSTM based Approach

In the previous methods, we were trying to predict different components of the opinion tuple like etarget and polarity. Our final goal is to predict the entire opinion tuple (source, target, polarity) for MPQA dataset, for explicit and implicit opinions. LSTMs (long short term memory networks) are ideal in this kind of scenario, as they can capture long-term dependencies of the input sequences in different layers. We referred to the experiments done in [1] for explicit opinions and extended the model with the hope of obtaining improvements in results. This section was done jointly with Arzoo.

GOAL : The goal is to predict a tag (BIO tag as discussed earlier) for each word in a given sentence in order to identify the source, target and opinion expression. In addition to that, the goal also involves predicting distances of the words that the current word is related to, on its left and its right.

METHODOLOGY : For every word w in the sentence, we output a tag (this is the BIO tagging and can be B_src, I_src, B_DSE, I_DSE, B_target, I_target, O) and left and right distances of related words (distance is measured by the number of tokens between the two related words). The maximum distance parameter was set to 15 in [1] but we decided to find the distance in each case in the gold annotations, experiment with it and then take the average. We got most optimal

results when this parameter was set to 10 for MPQA 2.0 dataset. The dimension of the output space depends on the number of possible tags and maximum distance value. For predicting the polarity jointly, one can add one more dimension to the output label space, which can be 0, 1 or 2 implying none, positive or negative polarity. We train the LSTM on several sentences to learn to predict the tag, left and right distances. These three components are exactly what we wanted to obtain, in order to generate the opinion tuple.

The LSTM Model

We referred to the model in [24] to build the LSTM here. The input gate, forget gate and intermediate cell state are computed as follows. Here x_t is the input to the memory cell layer at time t. W_i, W_f, W_c, W_o, U_i, U_f, U_c, U_o and V_o are weight matrices and b_i, b_f, b_c and b_o are bias vectors.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$\tilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

The new cell state can then be computed as follows:

$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1}$$

The hidden state and the output state would be calculated as follows:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o C_t + b_o)$$

$$h_t = o_t * tanh(C_t)$$

We use a multi-layer bi-directional LSTM as described in figure 3.4. For every token in a sentence, the hidden state h_t can be computed in forward as well as

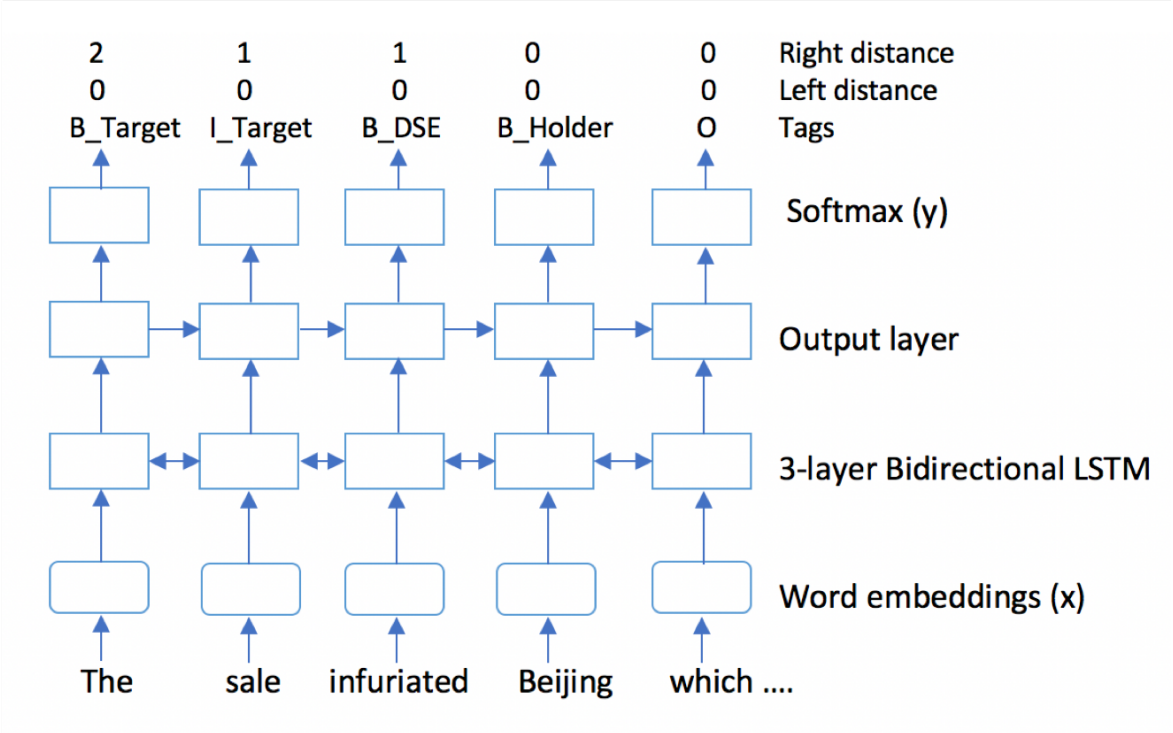

Figure 3.4: The LSTM Model

backward direction since it is bi-directional.

Word-Level Approach

As outlined in [25] by Collobert et. al., the score f_θ for the i^{th} tag given input x will be calculated by applying a softmax operation as given below. The numerator indicates a function of the likelihood that a particular word will have the predicted tag, and the denominator is for normalization, which is achieved by adding likelihood of all possible tags that the word can have.

$$p(i|x,\theta) = softmax([f_\theta]_i) = \frac{e^{[f_\theta]_i}}{\sum_j e^{[f_\theta]_j}}$$

Normalization is necessary as we are looking at probabilities. Taking log on both sides, we get the following normalized expression.

$$logp(i|x,\theta) \;=\; [f_\theta]_i - log(\sum_j e^{[f_\theta]_j})$$

This technique is called cross entropy and we use the relation level cross entropy error to train our network, as described below.

Sentence-Level Approach

In [1] and [25] the following equation had been used to calculate scores of sentence $[x]_1^T$ (the sentence has T tags) along a path of tags $[i]_1^T$ using the concept of Viterbi algorithm. It sums the transition scores and network scores for the particular tag at position t. A is the transition matrix here.

$$s([x]_1^T, [i]_1^T, \theta) = \sum_{i=1}^{T} ([A]_{[i]_{t-1},[i]_t} \;+\; [f_\theta]_{[i]_t,t})$$

Relation-Level Approach

In [1], the transition matrix A was introduced which stored transition scores for jumping from tag i_{t-1} and relation distance d_{t-1} to tag i_t and distance d_t. The score for a given sentence was calculated as follows.

$$s([x]_1^T, [i]_1^T, [d]_1^T, \theta) = \sum_{i=1}^{T} ([A]_{[i]_{t-1},[i]_t,[d]_{t-1},[d]_t} \;+\; [f_\theta]_{[i]_t,t})$$

We want to incorporate the scores for the tags of the words that this particular word is related to. Let us assume that the word at position t is related to a word to the left and to a word to the right by certain distance values l_t and r_t respectively. So the equation for our new model becomes as follows.

$$s_{left}([x]_1^T, [i]_1^T, [d]_1^T, \theta) = \sum_{i=1}^{T} ([A]_{[i]_{t-1},[i]_t,[d]_{t-1},[d]_t} \;+\; [f_\theta]_{[i]_t,t} \;+\; [f_\theta]_{[i]_{t-l_t},t-l_t})$$

$$s_{right}([x]_1^T, [i]_1^T, [d]_1^T, \theta) = \sum_{i=1}^{T} ([A]_{[i]_{t+1},[i]_t,[d]_{t+1},[d]_t} + [f_\theta]_{[i]_t,t} + [f_\theta]_{[i]_{t+r_t},t+r_t})$$

Setting the distance parameter

In the LSTM model, along with the BIO tag for each token, we predict the left and right distances of tokens that this token may be related to. We have to set a maximum parameter for this distance values, since the dimension of the output space depends on this. From the training data of MPQA 2.0, we observe that 93.6% of the sentences have distances less than or equal to 10, for the words they are related to and 97.5% of the sentences have distances less than or equal to 15. Now the choice of this parameter is a tradeoff between trying to cover more sentences and keeping the dimension of the output label space low. We set it to 10 after experimentation.

Hyperparameters

There are different parameters in this network and we experimented with different values for each of them before setting them to the one which gives better results. We experimented with epoch sizes ranging from 20 to 100, dropout was set to 0.5, window size for maximum distance was set to 10 after experimenting with values between 10 and 15, dimension for output of hidden unit was set to 50 and optimizer algorithm used was adadelta (stochastic gradient is another option).

Implementation Details

The implementation of this neural network was done using the standard functions available in the theano package of python.

Experiment 1

- Model : We used three hidden layers and an epoch size of 20. We had 6281 training sentences, 2298 sentences for validation and 892 for testing purposes.
- Results : For opinion holder, the precision is 0.566, recall is 0.695 and F1 score is 0.624, and for the target, precision is 0.51, recall is 0.428 and F1 score is 0.466 approximately. The code ran for 20 epochs, and the training took 271758.4 seconds.

Experiment 2

- Model : We used an epoch size of 100 for the previous model.
- Results : The code ran for 100 epochs, with 15624.361 seconds per epoch and training took 1562436.1 seconds. The results obtained previously in [1] are shown below in table 3.5 and our results are outlined in table 3.6.

Category	Overlap	Precision	Recall	F1
Opinion Holder	Binary	0.6275	0.6717	0.6471
	Proportional	0.5944	0.6551	0.6218
Opinion Expression	Binary	0.7173	0.7092	0.7111
	Proportional	0.6548	0.6554	0.6556
Opinion Target	Binary	0.6452	0.6594	0.6484
	Proportional	0.5275	0.6054	0.5581
IS-FROM	Binary	0.6419	0.5375	0.5822
IS-ABOUT	Binary	0.6248	0.4980	0.5498

Figure 3.5: Previous results from [1]

Category	Overlap	Precision	Recall	F1
Opinion Holder	Binary	0.5866	**0.7195**	0.6463
	Proportional	0.5777	**0.6949**	**0.6309**
Opinion Expression	Binary	0.6844	0.7062	0.6951
	Proportional	0.6259	0.6729	0.6486
Opinion Target	Binary	0.5290	0.6637	0.5888
	Proportional	0.3814	0.6313	0.4755
IS-FROM	Binary	0.6366	**0.5840**	**0.6092**
IS-ABOUT	Binary	0.5367	0.4721	0.5023

Figure 3.6: Results obtained in Experiment 2

Observation from Results

We introduced a new term while calculating the score for the predicted tag and distance for any token as outlined above. By comparing it with existing results in [1], we notice that the precision-recall values did not improve much. The reason behind this is that the dimension of the output space for the neural network is already quite high, since we are predicting tag and distance together. It is basically the number of different tags times the number of different distances (seven tags and eleven distance values possible). So an attempt to give more information by incorporating the scores for related words, is probably confusing the network in cases where the distance prediction is not correct. However, this method might improve scores if we had more sentences to train on, since performance of a neural network depends largely on the size of the training dataset. Also another interesting observation is that the LSTM based approach gave the best results in terms of precision, recall and F1 score values among all the methods that we tried.

CHAPTER 4

CONCLUSION AND FUTURE DIRECTION

For the first task, we proposed a bidirectional multilayered LSTM model for detection of sentiment, and got satisfactory F1 scores. This model can be extended in future for belief detection and other related problems as well, like relation classification. We alleviate overfitting by using dropout strategies. Experimentally, we found out that the UNK parameter was an important one, because changing it by too much affected the results which proves that handling of unknown words is a key factor. The epoch size gives saturation after a certain limit, due to memory bounds. We also compared our model with previous approaches. One can experiment further with the batch size next to see if it improves precision. Other ideas for improvement are to add more layers to the network, train on bigger datasets, use one-hot vectors for optimization, experiment with doc2vec instead of word2vec for word embeddings and try dropout embeddings instead of the traditional dropout strategies we used.

For the second task, we explored simple baseline approach, SVMs, CRFs, feedforward neural networks and LSTMs to detect opinions in the MPQA dataset. We also explored bootstrapping for improving confidence of our results but it did not help much. We wanted to explore random forests but it is tricky to select features for the decision trees. In future one can experiment further with this dataset using decision trees. Also we used theano for the neural network results on MPQA 2.0 but tensorflow might be more efficient. The LSTM based model can also be extended to the MPQA 3.0 dataset. Another idea for future work is to explore statistical script learning methods to predict the tuples for the implicit opinions and also try this for doing opinion inferences.

APPENDIX A

GLOSSARY OF TERMS

- **Neural networks :** As per Wikipedia, artificial neural networks or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming.
- **Feedforward neural network :** As per Wikipedia, a feedforward neural network is an artificial neural network wherein connections between the units do not form a cycle. The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves only in one direction which is forward, from the input nodes through the hidden nodes to the output nodes. Hence, there are no cycles or loops in the network.
- **LSTM :**Long Short Term Memory networks, usually just called LSTMs, are a special kind of recurrent neural network capable of learning long-term dependencies. They were introduced by Hochreiter Schmidhuber in 1997.
- **Conditional Random Field :** As per Wikipedia, conditional random fields are a class of statistical modeling method often applied in pattern recognition and machine learning and used for structured prediction. CRFs are a type of discriminative undirected probabilistic graphical model. It is used to encode known relationships between observations and construct consistent interpretations. It is often used for labeling or parsing of sequential data.
- **Integer Linear Programming :** ILP is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be

integers, implying that the objective function and the constraints (other than the integer constraints) are linear.

- **SVM :** Support vector machines as per Wikipedia are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.
- **Probabilistic Soft Logic :** Probabilistic Soft Logic (PSL) is a variation of Markov Logic Networks, which is a framework for probabilistic logic that employs weighted formulas in first order logic to compactly encode complex undirected probabilistic graphical models.

BIBLIOGRAPHY

[1] Katiyar and Cardie, "Investigating LSTMs for Joint Extraction of Opinion Entities and Relations," *Proceedings of the 54th ACL international conference*, vol. 54, 2016.

[2] Deep Learning, "https://en.wikipedia.org/wiki/Deep_learning."

[3] The 2016 TAC KBP BeSt Evaluation, "https://tac.nist.gov/publications/2016/additional.papers/TAC2016.KBP_Belief_and_Sentiment_overview.proceedings.pdf."

[4] Hochreiter and Schmidhuber, "Long Short Term Memory," vol. http://www.bioinf.jku.at/publications/older/2604.pdf, 1997.

[5] C. Bahdanau and Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," vol. https://arxiv.org/abs/1409.0473, 2014.

[6] M. Graves and Hinton, "Speech Recognition with Deep Recurrent Neural Networks," vol. https://arxiv.org/abs/1303.5778, 2013.

[7] Hammerton, "Named entity recognition with long short-term memory," vol. https://dl.acm.org/citation.cfm?id=1119202, 2003.

[8] Yang and Cardie, "Extracting opinion expressions with semi-markov conditional random fields," vol. https://www.cs.cornell.edu/home/cardie/papers/EMNLP13-Extracting.pdf, 2012.

[9] Irsoy and Cardie, "Opinion mining with deep recurrent neural networks," vol. https://www.cs.cornell.edu/oirsoy/files/emnlp14drnt.pdf, 2014.

[10] Understanding LSTM networks, "http://colah.github.io/posts/2015-08-Understanding-LSTM/."

[11] Deep MNIST for experts, 2016., "https://www.tensorflow.org/get_started/mnist/pros."

[12] Deng and Wiebe, "Joint Prediction for Entity/Event-Level Sentiment Analysis using Probabilistic Soft Logic Models," *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, vol. EMNLP-2015, 2015.

[13] The MPQA dataset, "http://mpqa.cs.pitt.edu/corpora/mpqa_corpus/."

[14] Deng and Wiebe, "MPQA 3.0: Entity/Event-Level Sentiment Corpus," *Proceedings of NAACL-HLT, 2015*, vol. NAACL-2015, 2015.

[15] Deng, "Entity/Event-Level Sentiment Detection and Inference," *Proceedings of NAACL-HLT Student Research Workshop, 2015*, vol. NAACL-2015, 2015.

[16] C. Deng and Wiebe, "Benefactive/Malefactive Event and Writer Attitude Annotation," *Annual Meeting of the Association for Computational Linguistics 2013*, vol. http://people.cs.pitt.edu/~wiebe/pubs/papers/acl2013.pdf, 2013.

[17] Yang and Cardie, "Joint Inference for Fine-grained Opinion Extraction," *Proceedings of the. 51st Annual Meeting of the Association for Computational Linguistics*, vol. http://www.aclweb.org/anthology/P13-1161, 2015.

[18] Pichotta and Mooney, "Using Sentence-Level LSTM Language Models for Script Inference," vol. http://www.cs.utexas.edu/users/ml/papers/pichotta.acl16.pdf, 2016.

[19] Pichotta and Mooney, "Statistical Script Learning with Recurrent Neural Networks," *Proceedings of the Workshop on Uphill Battles in Language Processing (UBLP)*, 2016.

[20] Pichotta, "Advances in Statistical Script Learning," vol. http://www.cs.utexas.edu/users/ml/papers/pichotta.thesis17.pdf, 2017.

[21] Pichotta and Mooney, "Learning Statistical Scripts with LSTM Recurrent Neural Networks," *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016.

[22] M. Lafferty and Pereira, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data," *Departmental*

Papers, vol. https://repository.upenn.edu/cgi/viewcontent.cgi?article=1162&context=cis_papers, 2001.

[23] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, "https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/."

[24] LSTM Networks for Sentiment Analysis, "http://deeplearning.net/tutorial/lstm.html."

[25] B. K. K. Collobert, Weston and Kuksa, "Natural Language Processing (Almost) from Scratch," *Journal of Machine Learning Research*, vol. http://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf, 2011.